NASA Contractor Report 172289

NASA-CR-172289
19840009831

ICASE

MODELLING ALGORITHM EXECUTION TIME ON PROCESSOR ARRAYS

Loyce M. Adams

and

Thomas W. Crockett

Contract Nos. NAS1-17070, NAS1-17130 (ICASE)
NAS1-16000 (Kentron Technical Center)

January 1984

INSTITUTE FOR COMPUTER APPLICATIONS IN SCIENCE AND ENGINEERING
NASA Langley Research Center, Hampton, Virginia 23665

Operated by the Universities Space Research Association

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23665

ENTER:
27 1 1 RN/NASA-CR-172289
DISPLAY 27/6/1
84N17899*# ISSUE 8 PAGE 1211 CATEGORY 61 RPT#: NASA-CR-172289
ICASE-84-1 NAS 1.26:172289 CNT#: NAS1-17070 NAS1-17130 NAS1-16000
84/01/00 26 PAGES UNCLASSIFIED DOCUMENT
UTTL: Modelling algorithm execution time on processor arrays TLSP: Final Report
AUTH: A/ADAMS, L. M.; B/CROCKETT, T. W. PAA: B/(Kentron Technical Center)
CORP: National Aeronautics and Space Administration. Langley Research Center, Hampton, Va. AVAIL.NTIS SAP: HC A03/MF A01
MAJS: /*ALGORITHMS/*ARRAYS/*COMPUTATION/*COMPUTER SYSTEMS PERFORMANCE/*PARALLEL PROCESSING (COMPUTERS)/*RESPONSE TIME (COMPUTERS)
MINS: / ARITHMETIC/ CONJUGATE GRADIENT METHOD/ INTERPROCESSOR COMMUNICATION/ PROBLEM SOLVING/ SYNCHRONISM
ABA: Author

Modelling Algorithm Execution Time on Processor Arrays

by

Loyce M. Adams
Institute for Computer Applications in Science and Engineering

Thomas W. Crockett
Kentron Technical Center

Abstract

A model for the execution time of parallel algorithms on processor arrays is described. The model is validated for the conjugate gradient algorithm on the eight processor Finite Element Machine at NASA Langley Research Center. Model predictions are also included for this algorithm on a larger array as the number of processors and system parameters are varied.

Research supported by the National Aeronautics and Space Administration under NASA Contract Nos. NAS1-17070 and NAS1-17130 while the first author was in residence at ICASE and by NASA Contract No. NAS1-16000 while the second author was employed at Kentron Technical Center, NASA Langley Research Center.

Introduction

The usual measure of the performance of a numerical algorithm on a sequential computer is the number of arithmetic operations required for completion of the algorithm. This number is generally expressed as a function of the size of the application problem. For instance, it is well known that the solution of a system of n linear equations using Gaussian elimination takes O(n3) arithmetic operations [1]. On sequential machines, the assumption is made that the execution time of the algorithm will be directly proportional to the arithmetic operation count. This approach has worked well as a means of comparing sequential algorithms.

However, it has been pointed out repeatedly in the literature [2,3,4,5] that this standard arithmetic complexity analysis is not sufficient to analyze parallel algorithms. Additional operations such as data transmissions between processors, synchronization of processors, and global decision-making may add to the execution time of the algorithm. The number of these operations and the time required per operation may vary with each algorithm, with the number of processors used to solve the problem, and with the problem size. The time for these operations is dependent not only upon the particular architecture in use, but also upon the software required to implement these operations. These considerations suggest that an adequate analysis of the performance of a parallel algorithm must include a model for its execution time.

This paper describes an approach for modelling the execution time of algorithms on parallel arrays as a function of the number of processors and particular system (hardware and software) parameters. The general characteristics of an execution time model are first described and then we show

how to apply the model for a specific algorithm on an actual processor array. We next describe a procedure for validating this model and then show how the model can be used to predict algorithm performance on larger processor arrays with different system parameters.

Performance Model

The execution time of an algorithm running on multiple processors that are cooperating to solve a given problem is the elapsed time from when the first processor begins execution to when the last processor finishes. For example, Figure 1 shows two processors working together to solve a problem. The solid line indicates busy time and the dotted line represents idle time.

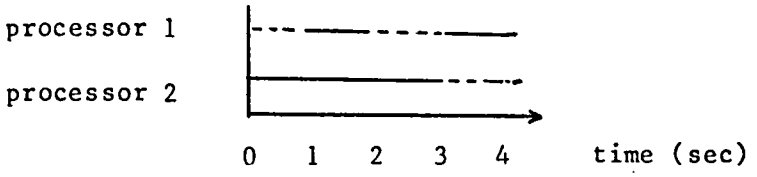

Figure 1. Cooperating Processors

The total execution time of the algorithm is 4 seconds even though processor 1 is busy for 2 seconds and processor 2 is busy for 3 seconds. By defining the execution time of the algorithm on a given processor to be the sum of the busy and idle times of that processor, the execution time will be the same for all processors. However, the execution time on all processors may need to be modelled if the interaction of the processors influences the components of the execution time.

The execution time (E) of a parallel algorithm can be broken down into four broad categories: arithmetic time (A), communication time (C), synchronization time (S), and idle or wait time (W) as shown in equation (1).

$$E = A + C + S + W \tag{1}$$

This model assumes one process per processor and no overlap of arithmetic, communication, and synchronization times.

The value for A includes the time for floating point operations, integer arithmetic, loop overhead, and array indexing. These integer operations are rarely included in sequential algorithm analysis but may be a function of the number of processors as well as the problem size in the parallel environment and should be included.

The value for C includes the time to communicate values from one processor to another. If the processors are connected to a shared memory, this time will be realized as the time to read and write into the shared memory. If, instead, the processors communicate directly with each other by passing messages over communication links, this time will be the time to send information to as well as receive information from cooperating processors.

The value for S includes the time the processor spends synchronizing with other processors and participating in global decision making. Synchronization followed by global decision making is necessary, for example, in parallel iterative algorithms. Each iteration, all processors must synchronize to determine if a global stopping criterion is met. For proces-

sors connected to a shared memory, this represents the time for all processors to write and then read a shared variable. If the processors are connected to special hardware for this purpose, this is the time for the hardware and its software interface to perform the operations. For some hardware array designs, synchronization and global decision making may be more appropriately modelled as communication time.

The value for W includes the time the processor spends waiting on values from other processors to arrive if communication links are used, or the time contending for shared memory if the processors communicate in that fashion. Also included in W is the time a processor is idle either prior to or following the execution of a task as shown in Figure 1.

In the next section, we show how to apply this model to predict the execution time of an iterative algorithm on NASA Langley's Finite Element Machine (FEM). The algorithm will first be described and the number of each operation summarized. The architecture and system software for the FEM is then discussed and the parameters of this machine that lead to the determination of A, C, and S in the model will be given.

Example Iterative Algorithm

The conjugate gradient algorithm [6] for solving a system of linear equations that arise from the discretization of a partial differential equation problem domain was chosen to illustrate how the number of operations in equation (1) are determined. A rectangular domain is discretized into computational nodes as shown in Figure 2 with an x by y block of nodes

assigned to each processor in the processor array.

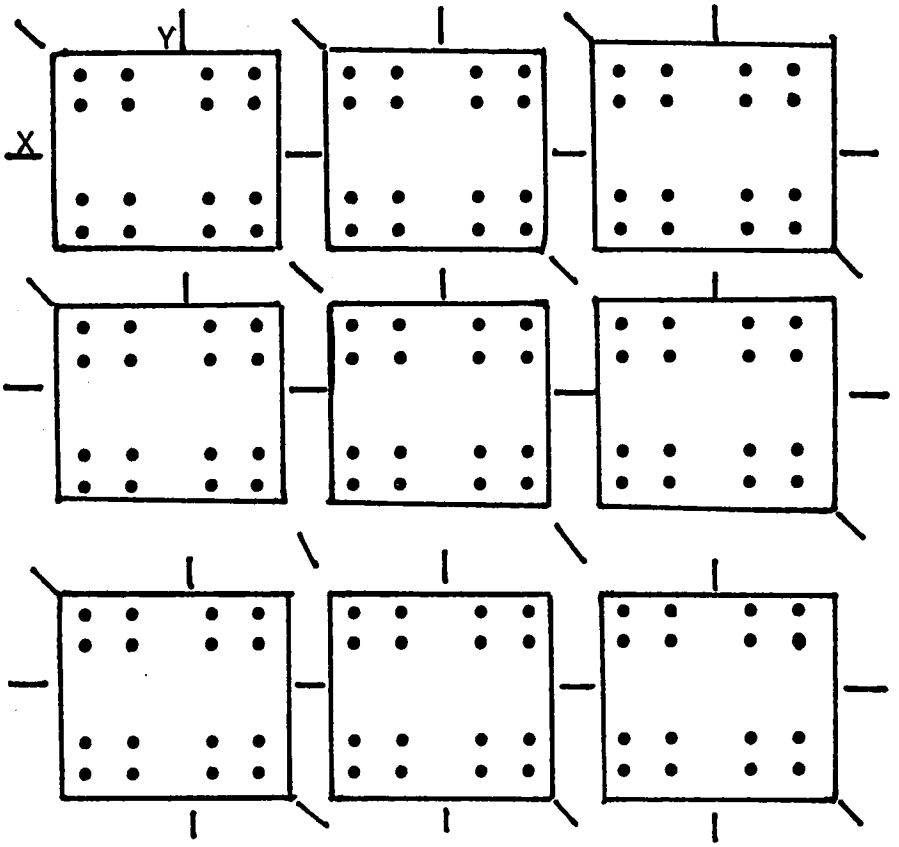

Figure 2. Problem Assignment to Processors

For this example problem suppose that each computational node contributes two equations to the system of linear equations to be solved. Each processor is therefore responsible for the solution of 2xy equations. If there are N equations total and p processors are used, the value of 2xy will be N/p. We now describe the components of the execution time for a processor in the interior of the processor array.

The parallel conjugate gradient algorithm for the problem described above can be shown [7] to require the following number of floating point

arithmetic operations. I represents the number of iterations of the algorithm.

multiply/add pairs	19NI/p
adds	2I(p-1)
divides	2I

(2)

On each iteration, two values for each computational node on the border of the x by y block must be sent to adjacent processors. Values going to the same processor may be sent as one or more packages depending on the structure of the particular architecture. Likewise, on each iteration, a processor must receive the two values for each border node of adjacent processors. Again, the values coming from a given processor may be received in one or more packages. In addition, the algorithm requires one value to be broadcast to the p-1 other processors and p-1 values to be received from the other processors twice each iteration. These communication components are summarized below where we assume that c numbers are sent per package and d numbers received per package:

packages sent:	4I(x+y+1)/c
numbers sent:	4I(x+y+1)
packages broadcast:	2I
numbers broadcast:	2I
packages received:	4I(x+y+1)/d + 2I(p-1)
numbers received:	4I(x+y+1) + 2I(p-1)

(3)

Each iteration, the processors must be synchronized and the global convergence test made. Therefore, the number of these components are:

$$\begin{aligned} &\text{number of synchronizations:} \quad I \\ &\text{number of convergence checks:} \quad I \end{aligned} \tag{4}$$

On sequential computers, the time for convergence checking is rarely counted in the complexity of the algorithm. However, this operation requires cooperation between the processors for the parallel algorithm and may be costly depending on the hardware and system software available to perform these operations.

The wait time for this algorithm is assumed to be zero since each processor in the interior of the processor array has the same amount of work to perform and the same code to execute. Unlike the situation shown in Figure 1, the processors will calculate, communicate, and synchronize more or less at the same time for this algorithm.

The times for each of the operations for the conjugate gradient algorithm depend upon the parallel array and its software. We address these factors for the Finite Element Machine in the following discussion.

The Finite Element Machine

The Finite Element Machine (FEM) [8,9,10,11] is a research computer being built at NASA's Langley Research Center to investigate the application of parallel processing to structural engineering analysis. FEM con-

sists of a minicomputer front-end, called the Controller, attached to an MIMD array of asynchronous microcomputers, referred to as the Array (Figure 3). Each processor in the Array is based on a TMS9900 microprocessor [12] with an associated Am9512 floating-point unit [13], 32K bytes of RAM, and 16K bytes of EPROM. Each processor has its own clock and runs its own program on its own data. There is no shared memory in the system. Additional circuitry provides a rich interconnection environment for communication and cooperative computation, which includes:

(1) Local communication links. Each processor has twelve bi-directional serial ports which provide dedicated I/O paths to neighboring processors. For this study, we choose the interconnection topology to be an eight nearest neighbor planar mesh leaving four links unused.

(2) Global bus. A 16-bit parallel time-multiplexed bus provides point-to-point and broadcast communications among all of the processors in the Array. The global bus also connects the Controller to the Array, and is used to load programs and data to the processors, and to retrieve results.

(3) Signal flags. Each processor has eight binary hardware flags which can be set to either True or False. Distributed circuitry allows each processor to inspect the global status of each flag. Status signals include ANY (one or more processors have set the flag to True), ALL (every processor has set the flag to True), and a special SYNC signal which is used for synchronization.

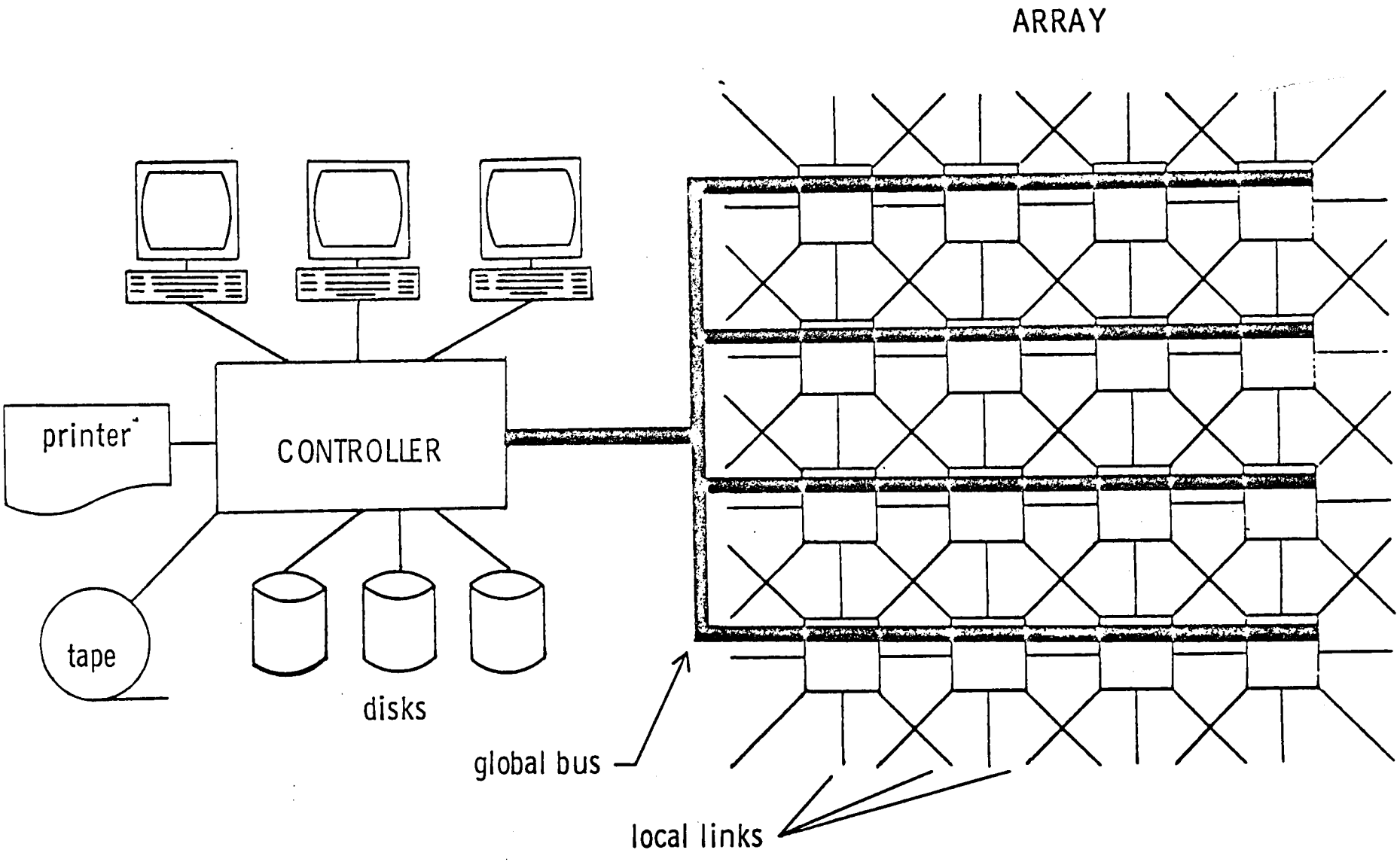

Figure 3. 16-processor Finite Element Machine.
(Flag network and sum/max tree omitted for clarity.)

(4) Sum/maximum network. The sum/max network aids in global calculations by determining the sum and maximum of the inputs from all processors [14]. This network is logically a tree with special-purpose computation nodes residing on each processor. The results of the computation are made available to all processors in the Array.

The current machine has eight processors operational (minus the sum/max network), and plans call for expansion first to 16, and eventually to 36, processors.

System software for FEM consists of three major components: (1) FEM Array Control Software (FACS), (2) Nodal Exec operating system, and (3) PASLIB subroutine library. FACS [15] is a set of about 40 programs which provide the user interface to the Array. Nodal Exec is a small special-purpose operating system which resides in EPROM on each processor in the Array. It is divided into two major sections, one of which provides the usual operating system support services such as memory management, I/O primitives, interrupt handling, and timing. The other section is a set of routines which carry out operations requested by the FACS software.

Application programs for the Array are written in Pascal. Programs are compiled and linked on the Controller, and downloaded to the appropriate processor(s) using FACS. Access to the special architectural features of the machine is provided by a library of subroutines called PASLIB [16].

Some PASLIB operations are implemented directly, while others are carried out through Nodal Exec. Services provided by PASLIB include communication with the Controller and other processors, flag and sum/max operations, mathematical subroutines, and timing.

Interprocessor communication is provided by three subroutines: SEND, SENDALL, and RECV. SEND transmits the requested number of data words at a given address to a particular neighboring processor. SENDALL is similar, but transmits the data to all neighboring processors. RECV accepts the requested number of data words from a neighboring processor and stores them at the given address. Nodal Exec provides buffering services so that communicating processors need not be tightly synchronized.

Model Parameters For FEM

Table 1 lists the execution time model parameters for the Finite Element Machine.

Cost Parameter	Time (milli-sec)
Multiplication/Addition Pair	0.997-1.195 (1.032 typical)
Addition	0.475-0.662 (0.516 typical)
Division	0.520-0.538 (0.526 typical)
Package Receive Overhead	1.308
Receive a Number	0.510
Package Send Overhead	1.672
Send a Number	0.221
Package Broadcast Overhead	1.672
Broadcast a Number	0.221
Synchronize	0.129
Global Flag Check	0.278

Table 1. FEM Model Parameters

In all cases, the software component of these operations dominates the hardware time required. Although proposed parallel architectures are frequently analyzed based on hardware arguments, our experience indicates that performance estimates can only be justified when the times for realistic software implementations are included. In many cases, accurate estimates of software overhead can only be obtained by writing the code and determining the execution time, either by summing instruction times, simulating execution at the instruction level, or running on the actual hardware.

The figures in Table 1 were obtained by either of two techniques, (1) adding up the instruction times required for the operation, or (2) performing timing experiments on the actual hardware. The time accounted for in each of these model parameters is discussed briefly.

The time for single precision floating-point operations on FEM are dominated by the time required to load operands and retrieve results from the Am9512, which has an 8-bit wide data path. Since actual computation time is operand dependent, a range of times for floating point arithmetic operations are given, along with a typical time.

Communication times include software overhead for parameter validation, table look-ups, buffering, and interrupt overheads. The times used here are based on a study of the PASLIB SEND and RECV routines done by J. Knott [17]. Broadcast times for FEM have not been measured, but are similar to those for non-broadcast SENDs. These times are subject to some variation caused by dynamic interactions among asynchronous processors.

The synchronization operation uses the SYNC flag signal to achieve synchronization among all processors. The synchronization value in Table 1 excludes any time which is spent waiting for slower processors to catch up. Actual wait times are dependent on the algorithm used and the variation in workload between processors; minor effects are caused by operand dependencies in floating-point operations and slight differences in clock speeds between processors. This is one of the most efficient operations implemented by PASLIB; most of the software time is attributable to subroutine call overhead.

Global flag status checks, like synchronization operations, are relatively efficient with subroutine entry and exit code constituting the major overhead. These are used by parallel iterative algorithms, for example, to determine global convergence.

Performance Measurement Tools

System software for FEM provides several tools which aid in the analysis of program behavior on the Array. These include timers, operation counters, and a statistical trace facility. Two interval timers are available on each processor. Nodal Exec uses one of them to measure elapsed program execution time, with a resolution of 16 milliseconds. This timer can be interrogated from user programs to yield the elapsed execution time at any point. The second timer can be started and stopped under control of the user program, and has programmable resolution from 1 to 349 milliseconds.

In addition, Nodal Exec and PASLIB maintain counters during program execution which record the number of floating-point and flag operations, send and receive calls, buffer allocations, I/O interrupts, and data words transmitted and received. This information can be retrieved by the Controller and post-processed to give an execution statistics report which includes not only the operation counts, but derived performance measures such as average input and output data rates, average number of data words processed per I/O interrupt, and overall floating-point rate. The operand counts provide accurate data to validate the numbers in the performance model, and the derived statistics give a rough idea of communication and floating-point efficiencies.

While the execution statistics can be used to validate the number of operations in the model, an additional tool is needed to validate the percentage of time spent in these operations and to pinpoint discrepancies between predicted and observed behavior. This capability is provided by Nodal Exec in the form of a statistical tracer. The tracer works by sampling the processor's program counter at regular intervals and sending the observed value to the Controller, where it is stored in a file. If enough samples are taken, the resulting distribution of program counter addresses will reflect the time spent in various portions of the program, including system code. The trace file saved by the Controller is sorted by processor number and program counter address, and the result can be correlated with the program's link map to give the approximate percentage of time spent in each routine. Since the trace resolution is at the instruction level, time spent in specific loops can be pinpointed.

Because the trace data is only a statistical sample, it is subject to some error, and trace results may vary somewhat from run to run. The recommended procedure is to make several runs of the same program and average the trace results. There is sufficient asynchrony between the trace mechanism (driven by a timer interrupt) and program execution that identical trace results from one run to the next have not been observed. The variance of trace results among several runs can be used to obtain a measure of confidence for the mean result.

Results

The conjugate gradient algorithm was run on an eight processor Finite Element Machine arranged as 2 rows and 4 columns of processors for two problems. For the first problem, each processor was assigned a 1x3 block of computational nodes; for the second problem, each processor had a 3x3 block of nodes. Each problem was run and the execution statistics matched exactly the number of operations predicted by the model. The problems were then run three times each with the tracer enabled using a 50 millisecond trace interval. The tracer increased execution time by about 0.9% and no noticeable effect on execution behavior was seen in the execution statistics. The results of the three runs were averaged and compared against those predicted by the model. Tables 2 and 3 summarize the findings.

	Tracer Results		Model Predictions
	Time(sec)	% Time	Time(sec)
Floating Point Arithmetic	8.96	56.7	8.57
Integer Arithmetic and Indexing	2.81	17.8	----
Communication and Synchronization	4.03	25.5	4.33

Table 2. 1x3 Block of Nodes/Processor

	Tracer Results Time(sec)	% Time	Model Predictions Time(sec)
Floating Point Arithmetic	34.93	67.3	34.44
Integer Arithmetic and Indexing	10.27	19.8	-----
Communication and Synchronization	6.70	12.9	6.28

Table 3. 3x3 Block of Nodes/Processor

Observe that the model predictions agreed closely with the trace results for the time required for the floating point arithmetic and communication and synchronization operations. The tracer also indicated that integer arithmetic, array indexing, and loop overhead should be modelled since this accounted for about 20 percent of the execution time in both problems.

This model can be used to predict the performance of the example iterative algorithm on larger processor arrays. As an illustration, assume that one number is sent and received per package and that the time to broadcast a package is equal to the time to send or receive a package. Furthermore, define the ratio of the communication time to arithmetic time, α, as below:

$$\alpha = \frac{\text{time to send a package}}{\text{time to do a multiply-add pair}}$$

With this definition, the execution time for our example algorithm can be expressed in terms of α and p, the number of processors. As an example, we consider the problem size to be fixed at a 16x48 block of nodes and let the number of processors vary from 1,4,16,64, to 256 (the block of nodes per

processor will vary from 16x48, 8x24, 4x12, 2x6, and 1x3 respectively). Figures 4 and 5 show the execution time of this problem as a function of the number of processors for machines with an α parameter of 10 and 1 respectively. Note that in both cases, there will be a point where adding more processors to solve the problem is not beneficial since the execution time will increase. The reason for this is that as the number of processors increases, the adds in equation (2) and the receives in (3) that are both $O(p)$ begin to dominate the execution time. To avoid this situation, these operations could be done with special hardware like the sum/maximum circuit on the Finite Element Machine which requires only $O(\log p)$ operations.

Conclusions

A model for the execution time of parallel algorithms on processor arrays has been presented which captures many of the additional complexity measures unique to a parallel environment. The model was applied to a parallel implementation of the conjugate gradient algorithm on NASA Langley's Finite Element Machine. Experiments were performed to compare the model predictions against actual behavior, and results showed that the floating point arithmetic, communication, and synchronization components of the parallel algorithm execution time were being modelled correctly. In particular, these results pointed out that the overhead caused by the interaction of the system software and the actual parallel hardware must be reflected in the model parameters.

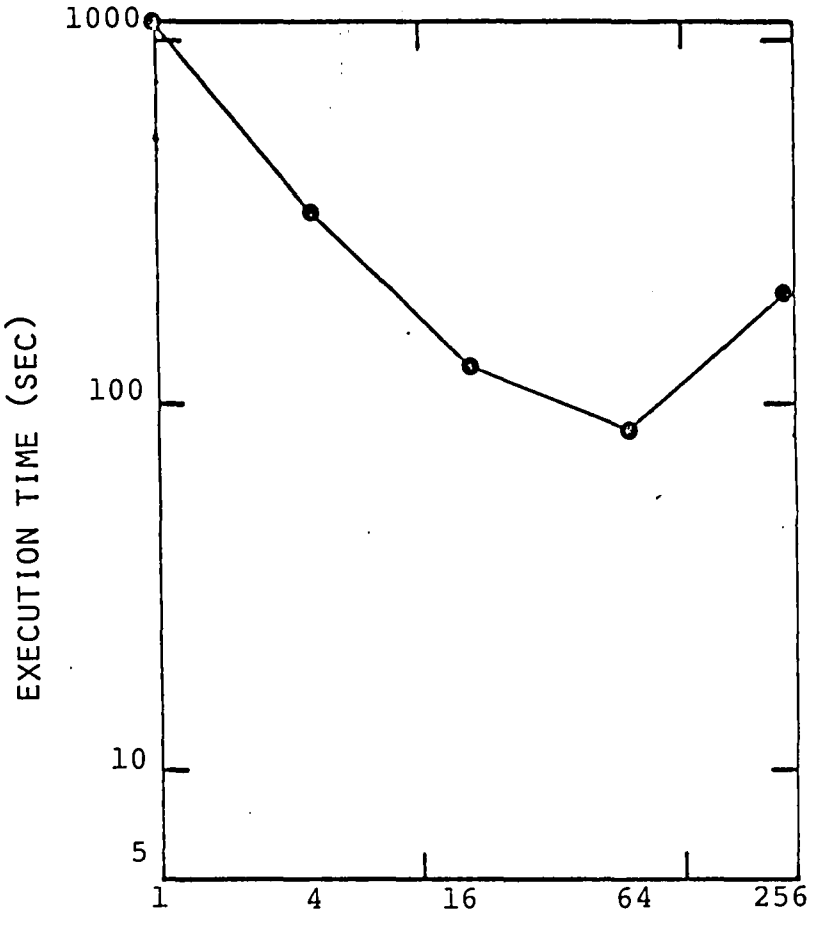

FIGURE 4. $a = 10$

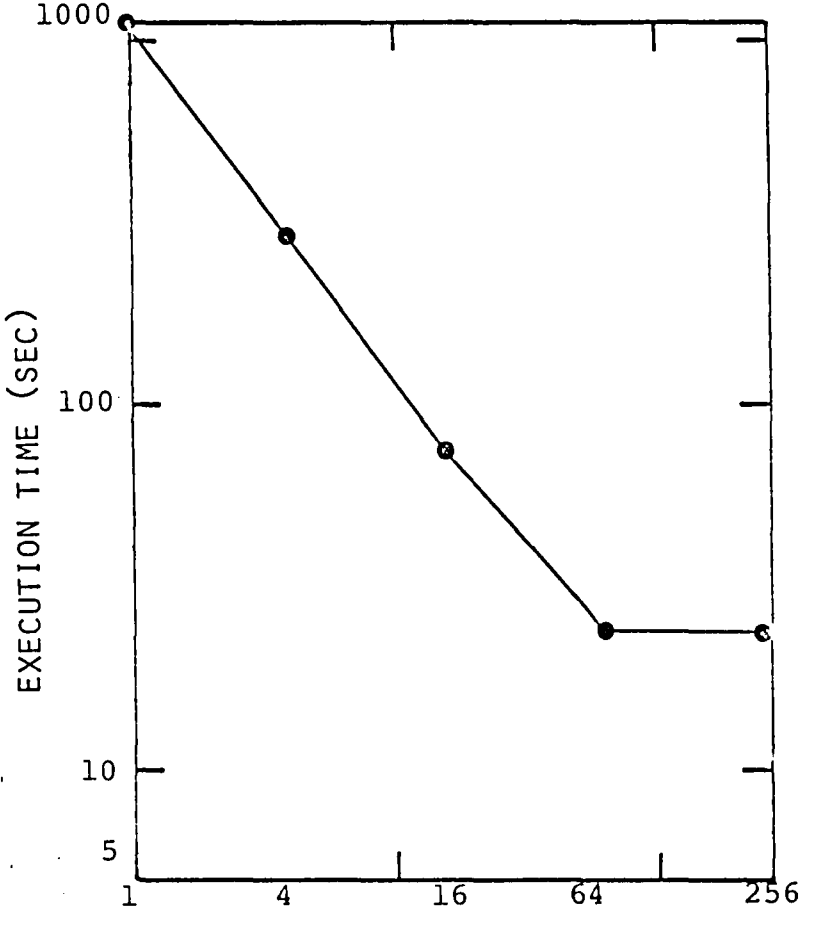

FIGURE 5. $a = 1$

The model was used to predict the performance of the conjugate gradient algorithm on a given problem as the number of processors and machine characteristics varied. In fact, the model can be used to address other issues such as algorithm speedup as a function of the number of processors; machine reliability as a function of the number of processors and machine parameters; algorithm comparisons as a function of problem size, number of processors, and machine parameters; and comparison of parallel and serial algorithms.

References

1. A. George and J. Liu, Computer Solution of Large Sparse Positive Definite Systems, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981, p. 21.

2. J. Ortega and R. Voigt, "Solutions of Partial Differential Equations on Vector Computers," Proc. 1977 Army Numerical Analysis Conference, pp. 475-526.

3. B. Buzbee, "Implementing Techniques for Elliptic Problems on Vector Computers," LA-UR 80-2343, Los Alamos Scientific Laboratory, Los Alamos, NM.

4. A. Jones and R. Schwarz, "Experience using Multiprocessor Systems- A Status Report," Computing Surveys, Vol. 12, No. 2, pp. 121-165.

5. R. Hockney and C. Jesshope, Parallel Computers, Adam Hilger Ltd., Techno House, Redcliffe Way, Bristol BS1, Great Britian, 1982.

6. A. Hageman and D. Young, Applied Iterative Methods, Academic Press, New York, New York, 1981, pp. 139-145.

7. L.M. Adams, "Iterative Algorithms for Large Sparse Linear Systems on Parallel Computers," Ph.D. dissertation, University of Virginia; also published as NASA CR-166027, NASA Langley Research Center, November, 1982, pp. 155-157.

8. H. F. Jordan and P. L. Sawyer, "A Multi-Microprocessor System for Finite Element Structural Analysis," Trends in Computerized Structural

Analysis and Synthesis, A. K. Noor and H. G. McComb, Jr., eds., Pergamon Press, Oxford, 1978, pp. 21-19.

9. H. F. Jordan, ed. "The Finite Element Machine Programmer's Reference Manual, CSDG-79-2, Computer Systems Design Group, University of Colorado, Boulder, 1979.

10. O.O. Storaasli, S.W. Peebles, T.W. Crockett, J.D. Knott, L.M. Adams, "The Finite Element Machine: An Experiment in Parallel Processing," Research in Struct. & Solid Mechanics, NASA Conf. Pub. 2245, Wash. D.C. 201-217, October 1982.

11. D.D. Loendorf, "Advanced Computer Architecture for Engineering Analysis and Design," Ph.D. dissertation, Department of Aerospace Engineering, University of Michigan, Ann Arbor, 1983.

12. TMS 9900 MICROPROCESSOR DATA MANUAL, Semiconductor Group, Texas Instruments Inc., Nov. 1975.

13. Am9512 FLOATING POINT PROCESSOR, Advanced Micro Devices, Sunnyvale, Ca., Apr. 1980.

14. H.F. Jordan, M. Scalabrin, W. Calvert, "A Comparison of Three Types of Multiprocessor Algorithms," Proc. 1979 Intl. Conf. on Parallel Processing, August 1979, pp. 231-238.

15. J.D. Knott, "FEM ARRAY Control Software User's Guide," NASA CR-172189, NASA Langley Research Center, Hampton, Va., Aug. 1983.

16. T.W. Crockett, "PASLIB Programmer's Guide for the Finite Element

Machine," NASA CR-172281 , NASA Langley Research Center, Hampton, Va., 1984.

17. J.D. Knott, "A Performance Analysis of the PASLIB Version 2.1X SEND and RECV Routines On The Finite Element Machine," NASA CR-172205, NASA Langley Research Center, Hampton, Va., Aug. 1983.

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA CR-172289		

4. Title and Subtitle	5. Report Date
Modelling Algorithm Execution Time on Processor Arrays	January 1984
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Loyce M. Adams* and Thomas W. Crockett**	84-1
	10. Work Unit No.

9. Performing Organization Name and Address		11. Contract or Grant No.
ICASE Mail Stop 132C NASA Langley Research Center Hampton, VA 23665	Kentron Technical Center 3221 N. Armistead Avenue Hampton, VA 23666	NAS1-17070, NAS1-17130, NAS1-16000
		13. Type of Report and Period Covered
		contractor report

12. Sponsoring Agency Name and Address	14. Sponsoring Agency Code
National Aeronautics and Space Administration Washington, D.C. 20546	

15. Supplementary Notes

Langley Technical Monitor: Robert H. Tolson
Final Report

*ICASE
**Kentron Technical Center

16. Abstract

A model for the execution time of parallel algorithms on processor arrays is described. The model is validated for the conjugate gradient algorithm on the eight processor Finite Element Machine at NASA Langley Research Center. Model predictions are also included for this algorithm on a larger array as the number of processors and system parameters are varied.

17. Key Words (Suggested by Author(s))

conjugate gradient
processor arrays

18. Distribution Statement

61 Computer Programming and Software

Unclassified-Unlimited

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	25	A02